Système de Chat Distribué (Client-Serveur) avec Java RMI

Description

Développez une application de chat distribuée où plusieurs clients peuvent envoyer et recevoir des messages via un serveur central en utilisant **Java RMI**. Chaque client se connecte au serveur, s'enregistre, puis peut échanger des messages en temps réel avec d'autres clients connectés.

Fonctionnalités requises

Côté Serveur :

1. **Gestion des connexions :**
    - Permettre aux clients de s'enregistrer en fournissant leur nom d'utilisateur.
    - Maintenir une liste des clients connectés.
2. **Transmission des messages :**
    - Recevoir un message d’un client et le distribuer aux autres clients connectés.
    - Envoyer des notifications lorsqu’un client rejoint ou quitte le chat.
3. **Déconnexion :**
    - Retirer un client de la liste des utilisateurs actifs lorsqu’il se déconnecte.

Côté Client :

1. **Connexion :**
    - Permettre au client de se connecter au serveur en saisissant un nom d'utilisateur.
2. **Envoi et réception de messages :**
    - Envoyer un message au serveur.
    - Recevoir en temps réel les messages provenant du serveur, y compris les notifications de connexion/déconnexion.
3. **Interface :**
    - Afficher une interface en console ou graphique pour saisir et afficher les messages.

Architecture

2 Tiers :

Le projet repose sur une **architecture 2 tiers**, où le client et le serveur interagissent directement :

1. **Tier 1 - Client :**
    - Les clients consomment les services du serveur.
    - Ils contiennent la logique d'interface utilisateur (UI) et une partie de la logique métier.
2. **Tier 2 - Serveur :**
    - Gère la logique métier principale et la coordination entre les clients.
    - Maintient une liste des clients connectés et relaye les messages.

Interfaces RMI

Interface Serveur :

Déclare les méthodes exposées par le serveur :

```
void registerClient(ClientInterface client, String username) throws
RemoteException;
void sendMessage(String username, String message) throws RemoteException;
void disconnectClient(String username) throws RemoteException;
```

Interface Client :

Déclare la méthode pour recevoir les messages depuis le serveur :

```
void receiveMessage(String message) throws RemoteException;
```

Étapes de mise en œuvre

1. **Créer l'interface RMI commune :**
    - Définir les méthodes côté serveur (`registerClient`, `sendMessage`, `disconnectClient`).
    - Définir la méthode côté client (`receiveMessage`).
2. **Implémenter le serveur :**
    - Maintenir une liste des clients connectés.
    - Diffuser les messages à tous les clients.
3. **Implémenter le client :**
    - Implémenter la méthode `receiveMessage` pour afficher les messages.
    - Fournir une boucle pour envoyer des messages au serveur.
4. **Tester :**

- Lancer le serveur.
- Connecter plusieurs clients et tester l'échange de messages.

Évolution vers une architecture 3 tiers

Pour introduire une base de données pour stocker l'historique des messages ou des utilisateurs, une architecture **3 tiers** peut être envisagée :

1. **Tier 1 - Client :**
    - Gère uniquement l'interface utilisateur.
2. **Tier 2 - Serveur d'application :**
    - Contient la logique métier et traite les requêtes des clients.
    - Gère les interactions avec la base de données.
3. **Tier 3 - Base de données :**
    - Stocke les données persistantes (utilisateurs, messages, etc.).

Dans ce cas, le serveur agit comme une couche intermédiaire entre les clients et la base de données, assurant une meilleure séparation des préoccupations.